AD-757 181

DATACOMPUTER PROJECT SEMI-ANNUAL TECHNICAL REPORT, FEBRUARY 1, 1972 TO JULY 31, 1972

Computer Corporation of America

Prepared for:

Army Research Office-Durham
Advanced Research Projects Agency

1972

DISTRIBUTED BY:

NTIS

National Technical Information Service
U. S. DEPARTMENT OF COMMERCE
5285 Port Royal Road, Springfield Va. 22151

AD757181

DATACOMPUTER PROJECT
SEMI-ANNUAL TECHNICAL REPORT

February 1, 1972 to July 31, 1972

Contract No. DAHC04-71-C-0011
ARPA Order 1731

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DDC
RECEIVED
MAR 12 1973
D

Reproduced by
NATIONAL TECHNICAL
INFORMATION SERVICE
U S Department of Commerce
Springfield VA 22151

Submitted to:

Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

Attention: Program Management

Computer Corporation of America

575 Technology Square
Cambridge
Massachusetts 02139

617-491-3670

DATACOMPUTER PROJECT
SEMI-ANNUAL TECHNICAL REPORT

[illegible] 1972 to July 31, 1972

Contract No. DAHC04-71-C-0011
ARPA Order 1731

Submitted to:

Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

Attention: Program Management

Computer Corporation of America
575 Technology Square
Cambridge, Massachusetts 02139

DATACOMPUTER PROJECT
SEMI-ANNUAL TECHNICAL REPORT

February 1, 1972 to July 31, 1972

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the U.S. Army Research Office-Durham under Contract No. DAHC04-71-C-0011. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

Table of Contents

Page

1. Overview..... 1
2. Hardware Installation..... 2
3. Software Design and Implementation..... 4
4. Coordination Activites..... 5
   4.1 Meetings and Conferences..... 5
   4.2 Weather Database Working Group..... 5

Appendix A: Working Paper No. 5, "Datacomputer Software Architecture--Revision 1", February 29, 1972..... 7

Figure

1. CCA Computer Installation: Block Diagram..... 3

1. Overview

The goal of the project is the development of a shared, large-scale data system for the ARPA community.

The system may be viewed as a box that performs the functions of data storage and data management on behalf of multiple computers simultaneously connected to the box.

The box contains a large-scale tertiary storage device, secondary storage (disks) for staging, and a medium-scale computer for performing data management functions.

Access to the box is through a device-independent notation, datalanguage. This language is being designed for use in the Arpanet as a standard means of access to remotely located data. It contains features specifically designed for sharing data among programs that operate on different machines, for describing a broad class of data structures, and for allowing arbitrary subsets of large files to be selected efficiently at run-time.

2. Hardware Installation

A PDP-10 system was delivered to CCA late in the last reporting period. This system was checked out, and regular DEC maintenance was begun in March. Also in March a BBN Model 701 Pager was delivered and integrated into the system. We are expecting delivery of a TIP early in the next reporting period. With the addition of the TIP, the installation will be as shown in Fig. 1.

System performance during most of this period was poor. Problems arose in various areas, but were centered on the ME10 core memories (these were new DEC products, replacing the better-debugged older MA10s) and the disk controller.

Towards the end of this period, DEC cooperated by providing on-site personnel daily and 24-hour-a-day on-call maintenance. Furthermore all outstanding ECOs were installed, and a major re-cabling effort took place.

A new policy of keeping power on for all units 24-hours-per-day was instituted. Subsequently, starting in July, performance began to improve markedly, and became satisfactory at the end of this period.

In regard to overall hardware system architecture, Working Paper No. 6, "Datacomputer Hardware Architecture", was completed and distributed. An activity aimed at evaluating existing tertiary storage devices was initiated; results will be given in the report for the next period.

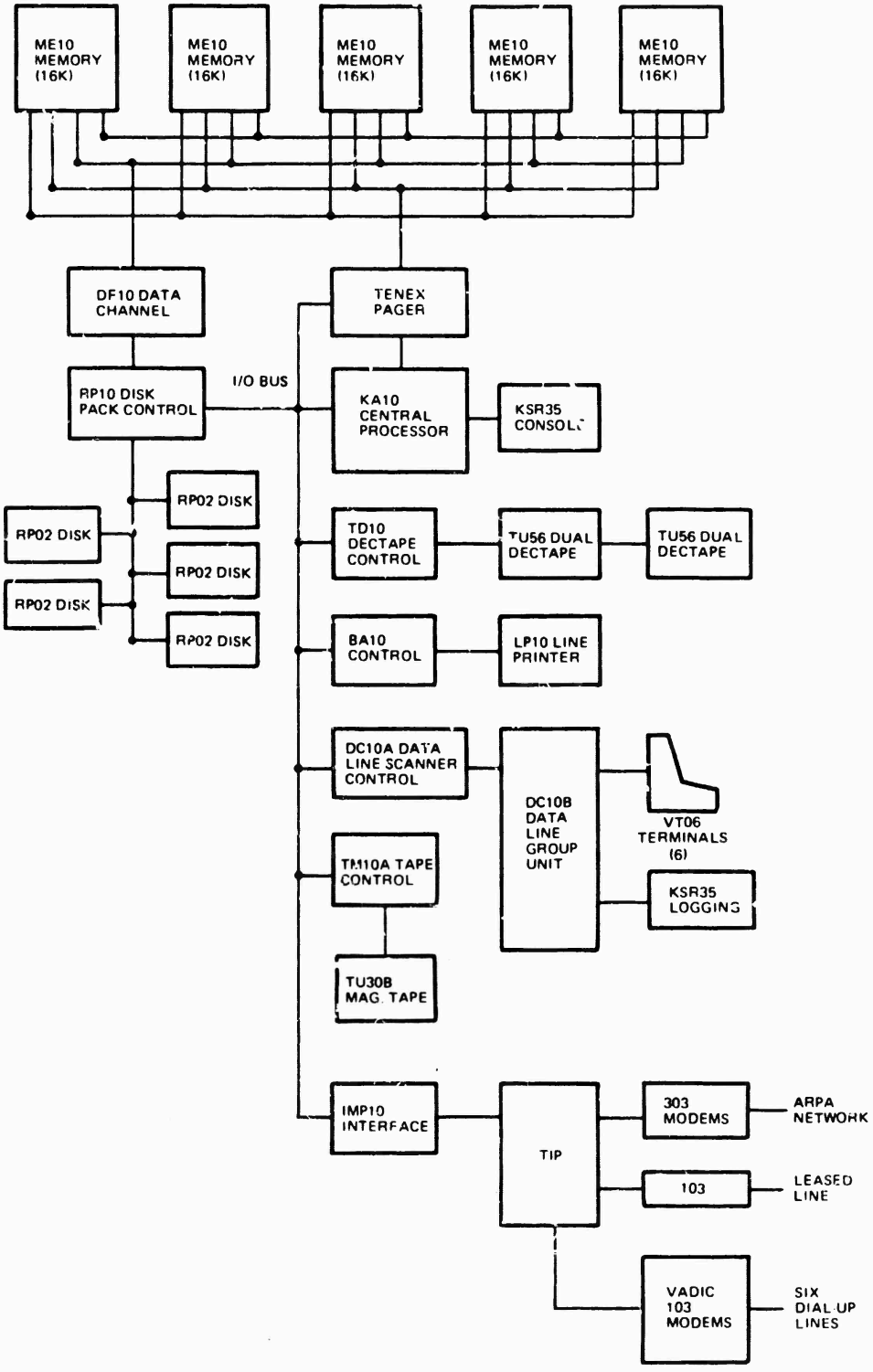

Figure 1

CCA Computer Installation: Block Diagram

3. Software Design and Implementation

During this period, the software system design reached a fairly stable state as documented in Working Paper No. 5, "Datacomputer Software Architecture--Revision 1", dated February 29, 1972, which is included here as Appendix A.

Regarding software implementation, there is as yet little progress to report. The immediate goal is the generation of a complete, though primitive, system in time to give a demonstration at the ICCC Conference in October. The results of this endeavor will be discussed in the report for the next period.

4. Coordination Activities

4.1 Meetings and Conferences

A substantial activity has developed during this period dealing with technical coordination with potential users of the datacomputer system, providing information to interested members of the computer science community, government, and industry. In addition to the work related to the Weather Database Working Group (see below), interaction took place with U. of Illinois (Center for Advanced Computation), NASA/Ames (Institute for Advanced Computation), RAND Corporation, NIH, National Library of Medicine, U. of Michigan, and DOT. A technical presentation on the project was given to the IEEE, Boston Section on May 23.

A significant conference was held at NASA/Ames on May 25 with representation from the Illiac IV project (M. Pirtle), ARPA (L.G. Roberts) and CCA (T. Marill). It was decided that the datacomputer software would run at NASA/Ames on a non-dedicated PDP-10/TENEX, using the installed UNICON 690 for tertiary storage. At CCA the system would continue to run on a dedicated machine, to offer backup for the Ames system through the network, and to offer a high-speed direct-connection option to Boston-area users.

4.2 Weather Database Working Group

The Weather Database Working Group (WDBWG) had been set up during 1971 with the mission of coordinating plans for the loading of the ETAC weather data base into the datacomputer system, for keeping the information up-to-date, and for providing access to interested groups.

The second meeting of WDBWG took place in Washington, D.C. on February 10 with participation from CCA as well as ARPA, RAND, ETAC, AWS, NCAR and NOAA. It was decided that the analysis and upper air files will be kept on-line. The mandatory surface data will be broken up into a set of chronological station files, each one of which will be one datacomputer file. Date, time, block and station numbers should be inverted.

Appendix A

Working Paper No. 5, "Datacomputer Software, Architecture--Revision 1", February 29, 1972.

Datacomputer Software Architecture

Revision 1

Datacomputer Project
Working Paper No. 5
February 29, 1972

Contract No. DAH[illegible] 1-71-C-0011
ARPA Order 1731

Computer Corporation of America
575 Technology Square
Cambridge, Massachusetts 02139

Preface

This paper discusses the concepts and the functional design of the datacomputer software. It is a revision of Working Paper 1 of this series, and presents a revised architecture. The most important change to the architecture is the definition of a fifth major system component: the directory system. A large number of minor changes have also been made, and the content of the paper has been reorganized.

Other papers in the series discuss the access language, the file structures, the hardware of the system, and related topics. Further papers will be issued from time to time. All papers are subject to revision without notice.

11

Table of Contents

		Page
Preface		i
Chapter 1.	Introduction	1
	1.1 The Datacomputer	1
	1.2 Datacomputer for the ARPANET	4
	1.3 Architectural Overview	6
Chapter 2.	The Request Handler	9
	2.1 Request Handler Function	9
	2.2 Request Handler Point of View	10
	2.3 Datalanguage	11
	2.4 Data Storage and Access Techniques	12
Chapter 3.	The Storage Manager	14
	3.1 Storage Manager Function	14
	3.2 Storage Manager Concepts	15
	3.3 Storage Manager Interface	18
	3.4 Mass Storage Device	22
Chapter 4.	The I/O Manager	25
	4.1 Function	25
	4.2 Inside Interface	26
	4.3 Outside Interface	28
	4.4 Internals	31
Chapter 5.	The Supervisor	34
	5.1 Function	34
	5.2 Design	35
	5.3 Implementation Strategy	36
Chapter 6.	Directory System	37
	6.1 Function	37
	6.2 The File Directory	39
	6.3 File Numbers	41
	6.4 Calls from the Storage Manager	42
	6.5 Calls from the Request Handler	43
	6.6 Datalanguage-Directory Interactions	44
	6.7 Restrictions	45

Chapter 1
Introduction

1.1 The Datacomputer

The datacomputer is a system which performs data storage and data management functions.

One may consider the datacomputer as a black box with multiple physical ports to which processors can be interfaced.

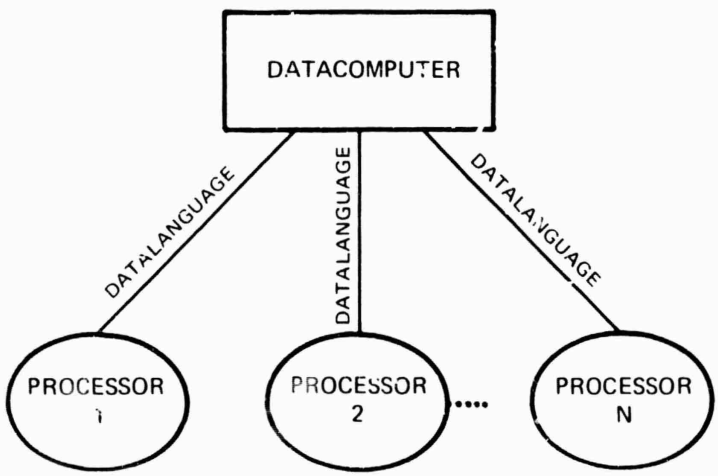

Figure 1.1 - The Datacomputer

Each of the processors can itself have multiple users, which can avail themselves of the services that the datacomputer offers.

Specifically, these services are:

1. On-line storage of data and data descriptions. A data file can be unusually large, up to one trillion bits (roughly the equivalent of 10,000 reels of magnetic tape.)

2. Retrieval of data (whole files, subsets of files, individual data elements).

3. File maintenance functions, that is, addition of new data, deletion of old data, changes to existing data.

4. Data reformatting.

5. Backup and recovery mechanisms, for use in case of failure in the datacomputer or in one of the user systems.

6. Accounting, for allocating charges to users.

7. Data sharing, allowing the same data bases to be accessed by different users.

8. Data privacy, preventing unauthorized access to data.

9. Simultaneous multi-user access, allowing more than one request to be serviced simultaneously.

A user program in an external computer interacts with the datacomputer only through datalanguage, a system of notation developed for this purpose. This increases the degree of integrity and privacy that can be achieved for the stored data, and improves the reliability of the system. It also allows users the convenience of working with a tool specifically designed for the job they are doing.

The datacomputer system is dedicated to data management and implemented on a large scale. Thus it offers more cost-effective and more extensive data management services than systems designed primarily for other purposes. Its hardware and software are specialized for the problems they most frequently encounter. On-line storage is orders of magnitude cheaper than in conventional systems. Data formats are flexible, and the variety in data structure is large.

1.2 Datacomputer for the ARPANET

The datacomputer for the ARPANET has two physical ports, as shown below:

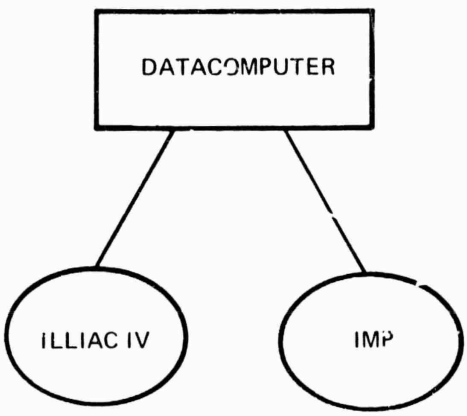

Figure 1.2 - the ARPANET Datacomputer

Here, the IMP is connected to the network and consequently allows a large number of processors to access the datacomputer through a low-speed (50,000 bits/second) port. Users of the ILLIAC IV have access at data rates several orders of magnitude higher than those available through the IMP.

Inside the datacomputer box are a modified PDP-10 processor, a BBN pager, several core memories, disks, interfaces to the IMP and ILLIAC, and a Precision Instruments UNICON 690 laser mass memory system. The UNICON contains three processors, two of which were built specifically for the storage system. The software for these processors is an integral part of the datacomputer software, and is outlined in Section 3.4. The UNICON has an on-line storage capacity of nearly one trillion bits. It also has the ability to mount and dismount storage packs of 25 billion bits, giving it unlimited off-line capacity on a low-cost medium.

This hardware configuration has been carefully designed and may be specialized further as the implementation continues. However, there is a level of design that is completely independent of the configuration. This includes the access language, most of the data storage, retrieval, and organization techniques, the interfaces of the five major modules, and their functional design. In addition, almost all of the software is independent of the mass memory system used. This point is particularly important, because mass memories are expected to improve considerably over the next few years. Thus additional mass storage devices can be accommodated with a minimum of reprogramming, and the entire configuration can be changed without loss of most of the design work.

1.3 Architectural Overview

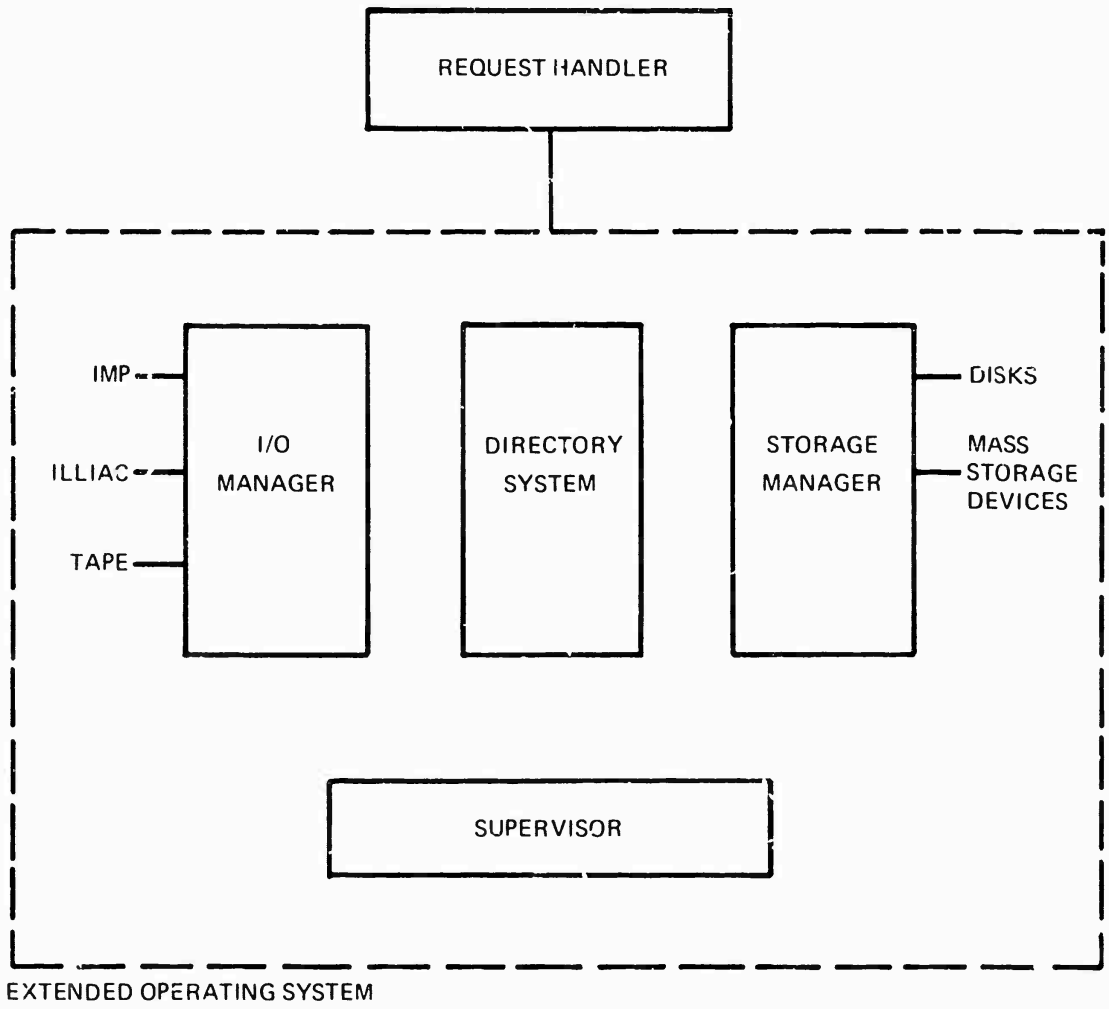

Figure 1.3 - Architectural Overview

The software of the datacomputer has five components: the request handler, the storage manager, the supervisor, the directory system, and the input-output manager (I/O) manager.

The storage manager, directory system, I/O manager, and supervisor comprise an extended operating system that supports one process (i.e., one job) for each user connected to the datacomputer. All of these processes execute the same program: the request handler. Each process acts independently, contending with the others for the resources of the system, and is concerned only with servicing its own user. The processes are started and stopped by the operating system and behave somewhat like user jobs in an ordinary multi-programmed computer.

While the request handler is concerned only with the user it is currently servicing, other datacomputer software modules are normally concerned with the entire system. The storage manager schedules for efficient use of the storage devices, at times degrading the service to one user while improving the service to others. The supervisor and I/O manager have scheduling functions which they carry out with a similar philosophy.

The functions of each of the five modules are described briefly below.

Request Handler

The request handler processes all datalanguage, including data descriptions; it structures data for storage and for output to users, determines and executes strategies for the accessing of stored data, and performs all data conversions.

Storage Manager

The storage manager controls the allocation of datacomputer storage and the scheduling of all storage device operations. It converts logical storage addresses to physical addresses, and converts data to and from physical storage formats.

Input-Output Manager

The input-output manager provides an interface between the request handler and the outside world. To the request handler, its services are: standard formats for all input and output data, standard error control, and standard control of connections with users. It accepts output from the request handler whenever it is generated, and accepts input from users when presented with it.

Supervisor

The supervisor schedules the use of the central processor, creates and deletes processes, and offers a variety of services normally associated with a multi-programmed operating system.

Directory System

The directory system catalogs data descriptions, file names and locations, file security information, and some accounting information.

Chapter 2
The Request Handler

2.1 Request Handler Function

All of the services offered by the datacomputer are implemented in the request handler. The user's datalanguage is processed and acted upon. Data is organized for storage, retrieved on demand, and formatted for output.

2.2 Request Handler Point of View

The request handler acts as though it were servicing a single user at a single, constant data rate, and working with a very simply and uniformly organized storage system. It is thus independent of system loading, buffering, scheduling, data rate, and device-peculiar data access considerations. It conceives data access strategies based only on the logical organization of data.

The request handler aids the other modules in their scheduling tasks by generating certain information for them. For example, it gives advance notice to the storage manager, when possible, of its requirements to access particular data. It also indicates convenient points for interruption, so that the system can efficiently switch to other tasks.

The request handler may conceive a sub-optimal strategy for a particular request, because it is relatively ignorant of the physical organization of the data. In this case, however, the system does not necessarily execute a sub-optimal strategy, since the storage manager has sufficient information to re-order the access operations. While the end result is not infallibly optimal for any particular request, it is globally more efficient than schemes that fully optimize each request with consequent loss of control at the system level. The present scheme also has the advantage of isolating the request handler from the considerations mentioned above.

2.3 Datalanguage

The activities of the request handler are best defined by datalanguage, its interface to the datacomputer user. In datalanguage, data is described and referenced:

A. so that the datacomputer can determine access paths and optimize the access process. Normally the user specifies the name or content of the data items desired, leaving the datacomputer to decide how to retrieve them. This is a convenience for the user who is not interested in the details of data structures. It is also an optimization for the usual case where the user is unaware of the actual location and storage format (for example, network users may frequently be unaware that their data is even on the datacomputer).

B. so that data sharing is facilitated. This is accomplished because data can be so thoroughly described that the datacomputer is aware of the format of the data in machine-independent terms. Thus the user can specify the format desired and leave the datacomputer to convert the data to this format if conversion is required.

C. in a concise, natural and convenient manner. This is accomplished primarily because the language need handle only data management problems, and is specialized for them (see Working Paper No. 3).

2.4 Data Storage and Access Techniques

While datalanguage allows users to program their own access techniques, most will prefer to use the ones built into the system.

Consider the problem of storing a large number of weather observations, say 500 million. Such a database might require 10 to 100 billion bits of storage. Even if the data is sorted on location and time, the problem of efficiently finding the data of interest in a particular problem is not simple. A typical datalanguage request for this file might ask for all the observations from a particular area showing high winds and warm temperatures. Without special data organization techniques, even if the area were small, one might expect to examine 1 million observations to extract the relevant ones.

There are only two ways to process such requests efficiently. One is to duplicate the data, sorting it different ways. This is impractical because of storage requirements. The second is to maintain an auxiliary body of information, that aids in answering the questions.

The organization of such a body, the application of it to retrieval, and the maintenance of it across changes to the data is the subject of a working paper "File Structures And Access Techniques". The basic technique employed is the use of inverted files, with extensions to accommodate files ordered by content. range retrievals, Boolean expressions, and tree-structured files. The techniques are relatively

insensitive to file size, and are expected to produce good results in the trillion bit range. Other common techniques, such as exhaustive sequential search, are used by the system when the more complex ones do not apply.

When the user requests the observations with high winds and warm temperatures, he need not be aware of the size of the database, the organization, or even the presence or absence of an inverted file. He defines in datalanguage the content and format of the desired data, and leaves the data computer to locate it the best possible way. Thus he is independent of all structure in the database except that which is basic to the use of the data, or that which affects performance so radically that it determines the user's behaviour in turn. This independence gives the datacomputer maximum freedom in organizing data.

Large, shared databases like the weather file described here will be reorganized from time to time as usage patterns and requirements alter. These reorganizations will be invisible, except for their effect on performance, to users that have been letting the datacomputer determine access strategies for them. Thus there are opportunities with the datacomputer to engineer global optimizations in the use of databases.

Chapter 3
The Storage Manager

3.1 Storage Manager Function

The datacomputer storage system includes core memory, conventional direct access devices, and a mass storage device. This system is complex, and its behaviour and configuration are subject to change during the life of the software. The storage manager has the function of using this system optimally, while presenting a stable and relatively simple interface to the request handler, directory system, and I/O manager.

3.2 Storage Manager Concepts

Storage manager users can operate on data only when it is in a buffer, a block of core locations of fixed size. Each user has a buffer table, containing a pointer to each of his buffers. All references to data in a buffer are indirect, through the buffer pointer. Users freely access buffers as though they were always in core. In practice, to optimize core usage, the contents of the buffers are continually being moved out to disk and back to core again. This movement of data, and the corresponding relocation of the buffer pointer, is invisible to the user.

All data are stored as updatable pages, which are blocks of data large enough to fill one buffer. Each page has a unique identifier, called the logical page address (LPA), and used to reference it in commands.

Via commands to the storage manager, users can create, delete, and access scratch pages and data pages. Scratch pages are used for storage of temporary and intermediate results that cannot conveniently be kept in the available buffers. Data pages are used for permanent storage of data.

When a scratch page or data page is read into a user's buffer, the original of the page in storage ceases to exist for most purposes. When the user modifies the contents of the buffer, he is now modifying the original and permanent copy of the page. In general, other requests for the page are satisfied by returning a pointer to the same copy. This scheme uses buffer space economically and eliminates unnecessary accesses to secondary storage. It also necessitates careful treatment of the data in the user's buffer, since it is logically the original of the page.

Consider the operation of extracting half of the data from a page, prefixing a count to it, and passing it to an output routine. This is most easily done by reading the page into a buffer, inserting the count at the proper position, and calling the routine with a pointer into the buffer. However, inserting the count into the buffer permanently alters the page, because the data in the buffer is the only and original copy of it. The relevant data could be copied into a second buffer, and there prefixed with a count. This undesirable solution creates two physical copies of the data in core, when typically only one is needed, and a third copy exists somewhere in storage.

This problem is solved by defining the <COPY> operation, which makes a private copy of the contents of a buffer. The copy of the data in the users buffer loses its identity as a scratch or data page. It now has the same status as random data placed by a user into a newly allocated buffer. If the buffer contains the only physical copy in the system of the data, then the <COPY> does the in-core copy-over required.

(Note: In practice, the association between private page and original copy is maintained until the private page is modified. This further optimizes the use of storage space and devices in certain cases, and is possible because of the implementation, which uses special paging hardware. In the general design, no such hardware is assumed, and the storage manager behaves as described here.)

The storage manager is expected to perform so that:

A. buffers declared by the user to be <IN-USE> are normally in core when the user is running.

B. other buffers are available after (at worst) short delay.

C. retrieval of scratch pages usually requires a short delay.

D. retrieval of data pages can usually be achieved with only short or medium delays, especially when intention to access them has been stated in advance, with a <PRE-READ> operation. (See below)

However, not all data pages are equally accessible. They are organized into files, and the initial access to a file may occasion a longer access time.

For the planned configuration, short delays are 10-200 milliseconds, medium delays are 200 milliseconds to 1 second, and long delays are 1-100 seconds.

3.3 Storage Manager Interface

The storage manager accepts the following commands from its users:

ALLOC <buffer number>

A buffer containing all zeroes is created and can be referenced by its buffer number in subsequent commands. Data in the buffer is effectively in core and can be operated upon directly by its owner.

REL <buffer number>

The buffer is disassociated from its contents, and the buffer number becomes available for re-use. Until the buffer is re-allocated, references to locations in it are in error. If the buffer contains a data or scratch page when released, that page must not have been modified while in the buffer.

DPR <buffer number, data page LPA, ROB>

SPR <buffer number, scratch page LPA, ROB>

A buffer containing the specified page is created, and can be referenced by the buffer number in subsequent commands. Data in the buffer is effectively in core and can be operated upon directly by its owner. The data can be modified if the read-only bit (ROB) is zero.

DPW <buffer number, data page LPA, ROB>

SPW <buffer number, scratch page LPA, ROB>

If the buffer contains the specified page, the buffer is disassociated from the page and released (see REL, above). If the buffer does not contain the specified page, then the buffer's contents replace the page. The buffer is then disassociated from its contents and released.

The read-only bit provides a means to request error-checking services from the storage manager. It can be set to one only in the first of the two cases explained above. When so set, it requests that an error condition be raised if the page has been modified. This is identical to releasing the buffer, except that it also insures that the page in the buffer has the LPA supplied in the command.

COPY <buffer number>

The contents of the buffer are disassociated from any scratch page, data page, or other buffer. The buffer's status is identical to that of a newly-allocated buffer, except that it contains the same data as it did before the COPY. Data in the buffer is effectively in core and can be operated upon directly by its owner. Subsequent changes to the buffer's contents will not affect the original source of the data.

ASSIGN/DEASSIGN <scratch page LPA or LPAs>

BLOCK

It is a convenient time for the user to be blocked. Thus, if the storage manager intends to block this user, the block should occur now. Before restarting the user (if indeed it was blocked), the storage manager would probably insure that the user's in-use buffers were in core.

These commands assign and deassign scratch pages. When a page has been assigned it may be used in a SPR or SPW command.

The following five commands are for the purpose of informing the storage manager of a user's intentions. They are not required, and are given only to aid the storage manager in making scheduling decisions.

PRE-READ <LPA>

The user intends to read the page sometime in the future and the storage manager should be prepared.

POST-READ <LPA>

The effect of a previous pre-read command for the page is negated. The user does not intend to access the page in the future.

IN-USE <buffer #>

The process intends to reference the indicated buffer. The storage manager will attempt to keep the page in core.

NOT-IN-USE <buffer #>

The effect of previous in-use commands is negated.

3.4 Mass Storage Device

The storage manager's main function is to manage the datacomputer storage system so that it is efficiently used, while the rest of the datacomputer system remains largely independent of storage hardware considerations. In this section a short discussion of how this is accomplished with the UNICON 690 is presented.

Recall that the storage manager users--the request handler and directory system--act as though data were stored on updatable pages of fixed size. The UNICON stores data in fixed-size units, clockwords, containing 64 data bits, which cannot be changed, once they have been written. The storage manager simulates updatable pages with the SOUPS scheme. With SOUPS, all pages are regarded as initially containing all zeroes. When a page is updated, a modification record is written. The modification record indicates the new contents and the location of any words changed. When a page is read from storage, all modifications are read in sequence, into an initially zeroed buffer. When all modifications have been processed, the buffer contains the latest copy of the page.

SOUPS requires that space be allocated for modification records, that too-frequently modified pages be rewritten in a new location, and that an entire clockword be written to change an isolated bit. In spite of these drawbacks, it is appropriate for a wide range of applications, and admirably hides the write-once property of the storage medium.

The response properties of the device (access times, rotation times, etc.) are more complex than those of a disk or drum. The volume of data storage is a strip, containing 1.7 billion bits. Strips can be mounted on either of two rotating drums, in which case they can be read or written, or they can be in the carousel, from which they can be mounted without human intervention. Worst case access time for data on a mounted strip is 500 milliseconds, while strip-changing time is as high as 10 seconds. Thus it is vital to minimize strip changes.

Recall now how the storage manager is to present the access properties of the storage system to its users. The set of all data pages is partitioned into logical units called files. Pages within a file are considered equally accessible. Since the request handler recognizes that crossing file boundaries should be minimized, it can help to minimize strip changes, without being tied to the concept of a strip or a strip mount.

Also, the storage manager can most effectively minimize strip mounts for the system by making use of PRE-READ information, which it gets from all its users, and by using its enormous disk buffer. For example, a process requesting input can run until it needs a page that is not available on disk or on the mounted strip, and can then block. When it is again started up, most of its pages will have already migrated to disk from the UNICON.

The PRE-READ system is least effective for processes that cannot determine the addresses of the pages they want without first looking at some other pages on the same strip. For this case, the strip-changing algorithm can compensate somewhat by not dismounting a strip until the process that requested it gets some opportunity to generate more requests. Another approach is to reflect in the PRE-READ information the intent to read more pages from the same file after the first request is satisfied.

To summarize, the UNICON, which is unusual in its response characteristics and storage medium, is accommodated in the storage manager with a special implementation of the idea of page, and a standard interpretation of the idea of file. The PRE-READ information and large buffer space further reduce the difficulties of hiding the device characteristics.

Chapter 4
The I/O Manager

4.1 Function

The I/O manager is the datacomputer's interface with the outside world. All data, and all requests for service, initially enter the datacomputer through the I/O manager. Thus on the outside, the I/O manager has the problem of interfacing with a variety of hardware devices, data formats, and software systems. On the inside, it services the datacomputer system by providing a standard protocol for connection control. It also buffers data when necessary, to make the rate of data flow convenient to datacomputer users and to the request handler.

4.2 Inside Interface

A caller can give four commands to the I/O manager: GET, PUT, OPEN and CLOSE. The caller is usually the request handler.

OPEN establishes a connection between the caller and a source of data in the outside world. The source can be a program in another computer, a file of data on tape, etc. The caller's end of the connection is a logical port. OPEN returns a port number, which the caller can use to reference the port established.

CLOSE terminates such a connection, and makes further references to the logical port invalid.

GET causes the I/O manager to pass one block of data or one item of control information to the caller.

PUT passes one block of data or one item of control information to the I/O manager from the caller.

The control information passed in GETS and PUTS is data stream punctuation. An example is the inter-record gap, when the physical tape record boundary has logical significance. Here the inter-record gap acts as punctuation, yet it is not data in the same sense as the data in the record. If the I/O manager reads an inter-record gap on a tape file, an item of control information is passed to the caller. If the caller wishes an inter-record gap to be written on a tape file, he must pass an item of control information to the I/O manager. (Note. This is true, of course, only when the inter-record gap has logical significance.)

e control information passing between I/O manager and caller is communicated by means of GETs and PUTs, and must be synchronized with the data stream itself. A GET or PUT passes either a block of data or a control information item. Thus a block of data can contain no control information. Aside from this, there are no restrictions on the size of the block, except implementation-dependent ones. For design purposes, a block can be defined as a convenient chunk of data containing no control items.

4.3 Outside Interface

The outside interface of the I/O manager is defined in terms of the datacomputer implementation. Initially, the datacomputer system will be interfaced to the ARPA network and the ILLIAC IV. In addition, the I/O manager will support magnetic tape as a means of data input/output, and local teletype-compatible terminals for debugging.

The software interface to the ILLIAC IV is under study at the time of this publication.

The software interface to the ARPA network, at the I/O manager level, is defined by the official network protocols, NIC 7104. Below, some especially relevant features of these protocols are discussed.

On the network, a user becomes connected to the datacomputer by executing the Initial Connection Protocol. As a result of this he has two half-duplex connections. On one he can send messages to the datacomputer, and on the other, the datacomputer can send messages to him. The messages are formatted, and message flow is controlled, according to HOST-HOST Protocol. Each message has a header and text. The header identifies the connection, states the length of the text, and supplies other miscellaneous information. The text contains an integral number of bytes of data, where byte size is a parameter determined during the initial connection procedure.

The boundaries of messages have no logical significance, so the concatenation of the texts of all the messages sent over a single connection constitutes a single string of bytes.

The Data Transfer Protocol is used to partition the byte stream into logical units. A byte stream formatted according to Data Transfer Protocol becomes a stream of transactions. There are several types of transactions, of which three are of interest here:

A. The data transaction, which is used to send all data.

B. The control transaction, which is used to send all datalanguage and diagnostics.

C. The information separator transaction, which is used to make logical unit boundaries.

To transmit a group of data records to the datacomputer, each record might be formatted as one or more data transactions. Following each record there would be an information separator, marking the end of the record. The end of the group would be indicated with a higher level separator.

Data Transfer Protocol allows the I/O manager to distinguish between datalanguage and data on the same connection, and to identify logical units and groups of logical units. The former ability is significant for detection of user errors and synchronization of user program and datacomputer recovery. The latter is useful for scheduling within the datacomputer.

Magnetic tape is viewed as a data I/O device. That is, controlling datalanguage is assumed to be coming from another source.

The tape medium has two kinds of information separators: inter-record gap and end-of-file marker. In addition, tape contents can have structure in the form of labels, recording formats, and even directories. Any such structures that are not to be described explicitly in datalanguage and processed by the request handler, must be "understood" by the I/O manager. To make the problem simpler at the outset, the I/O manager will handle a relatively small number of tape formats. Since most data is being transferred in or out over the network or over the ILLIAC IV interface, this restriction is deemed unimportant.

4.4 Internals

In terms of the system architecture, the I/O manager's most interesting problem is buffering. Except when the amount of buffered data for a certain connection becomes excessive, or when the system is dedicating too much space to I/O buffering, the I/O manager will accept all the input with which it is presented. This means:

A. to most users, the datacomputer appears to accept data as fast as they can send it

B. to the request handler, most users appear to accept data as fast as it (the request handler) can generate it.

C. to the request handler, some users appear to produce data at the rate of the datacomputer storage system.

These three effects are extremely desirable. For the user interfacing with the datacomputer at a data rate below that of the datacomputer storage system, the first is helpful because it enables him to transfer a given amount of data to the datacomputer in the minimum possible time. For moderate amounts of data, this will be independent of the amount of processing the datacomputer must do prior to storage of the data. Likewise, it will be independent of the loading of the datacomputer. (This effect is not achieved for the ILLIAC IV system, which can sustain data rates of one billion bits per second for up to one second. However, the I/O manager will accept input from the ILLIAC IV up to the

data rate of the datacomputer storage devices. Currently this is limited to 3.3 million bits per second, but studies have shown that there is a feasible way to raise this to 40 million bits per second for bursts up to several billion bits.)

The second and third effects are useful because of the way in which the datacomputer operates, and the conditions it will encounter. To service a particular user a certain set of buffers belonging to that user, the working set, must normally be in core. If any of these buffers, which are all frequently referenced, are not in core, then very little service will be given before the request handler will reference one that must be read in from disk. The working set is established dynamically, and migrates from disk to core when the user is getting some service. Optimal core usage is achieved when the request handler process gets its working set in core and then interfaces with the user at storage system rates. These rates are of course achieved when the I/O manager has queued some input or when the I/O manager is accepting and queueing all output at the rate the request handler generates it.

The buffering is implemented by queueing the data, by destination, initially in core and, when required, on scratch pages. Queueing on scratch pages involves an in-core copy operation to pack the data onto the pages, except when the unit being added to the queue is a full page of data. In this case the storage manager is asked to copy the page, which it can usually accomplish without the physical copy operation.

For some devices, reformatting of the data stream is required between useful request handler processing and the device I/O operation. An example is a tape drive transmitting 36-bit words (this is a problem because all request handler data is in 32-bit words). Here the I/O manager must do a shift-and-copy operation, which it will combine with formatting the data on a queue page, when that operation takes place. When no queue is forming, a copy is forced in this case. In the more normal cases, however, there are one or zero copy-overs in the I/O manager if the data gets queued, and none if it doesn't.

The actual response of the core management system in any real situation is determined dynamically according to the scheduling heuristics in the storage manager. The previous discussion is presented only to suggest the advantages that can be obtained by appropriate buffering in the I/O manager. Since both the size of queues and the management of all core is the function of the storage manager, the I/O manager is only concerned with formatting, building and maintaining the queues.

Chapter 5
The Supervisor

5.1 Function

The supervisor schedules the central processor, creates and deletes processes and performs some miscellaneous operating system functions. Among these are system bootstrap loading, operator communication, and management of the clock and priority interrupt system.

In scheduling the CPU, the supervisor determines:

A. When to interrupt a running process for the purpose of giving another process the chance to use the CPU. Many processes are designed to interrupt themselves at convenient times, but this is not always adequate.

B. Which waiting process should run next, when the last running process has stopped.

C. When there are too many or too few processes contending for the CPU and core. When such a determination has been made, the storage manager is informed and tries to rectify the situation by swapping processes in or out.

5.2 Design

Because the functions of the supervisor are so basic to any level of datacomputer system operation, the supervisor is more rigorously engineered, with respect to crash/recovery, than the other four modules. When any of the other modules crash, the supervisor obtains control and initiates recovery procedures. While occasionally a non-supervisor crash can have severe short-term consequences, such crashes are not generally as serious as supervisor crashes, in which reloading the entire system, human intervention, and (conceivably) loss of most or all temporary information is implied.

Thus extreme care is taken in allotting functions to the supervisor. It is constrained to be simple in design, to retain control under all conditions its designers can conceive of, and to delegate to other modules any functions not absolutely basic to its function.

5.3 Implementation Strategy

The supervisor will evolve during the implementation of the datacomputer. Initially it will consist of appropriate modules from an existing operating system. In this stage, scheduling algorithms may not be optimal for the datacomputer. When a primitive datacomputer has been established, design and implementation of the real supervisor will be undertaken. The design effort will then be based on some useful experience and on a better understanding of the datacomputer system than is possible presently.

Chapter 6
Directory System

6.1 Function

The directory system maintains data descriptions, file names and locations, file security information and some accounting information.

In some respects the directory system's functions are similar to the request handler's, and the directory system could probably be implemented in datalanguage. However, the directory system's database is of such crucial importance to the operation of the datacomputer, that it is separated from the request handler for the sake of reliability. The directory system can be expected to stabilize and undergo little change during the second half of the implementation of the request handler. During this period, it will be important to isolate the file directory from any bugs that occur in the request handler.

When the full datalanguage has been implemented, the request handler could indeed become as stable as the directory system. At this point, integration is undesirable for other reasons. For one thing, further development of the datacomputer concept may at this stage have implications for the directory system. With multiple datacomputers in the same system, data disposition among the datacomputers is something that should probably be settled by the datacomputers in a fashion invisible to their users. Under these circumstances, the directory systems of the several datacomputers must either cooperate or be combined into a single directory system. Inter-system problems should probably be handled by the

directory systems and storage managers and hidden from the request handlers.

Thus the directory system has different reliability requirements and will probably eventually have decidely different functions than the request handler.

6.2 The File Directory

Most of the directory system's data is stored in the file directory, and a major part of its job is the maintenance and use of this directory.

The datacomputer file directory is unusual in both size and content. Just the on-line storage will give rise to an extremely large file directory by conventional standards. (To get a feel for this, consider the on-line storage as the equivalent of 10,000 tapes.) The off-line storage is unlimited, and must be kept in the directory also. In content the directory is unusual because it contains not only file names and pointers, but datalanguage descriptions, which are used in the compilation process. The distinction between files and other kinds of data containers (like records, trees, fields, groups of records, etc.) is somewhat blurred in datalanguage, and this may give rise to heavier usage of the file directory than in other systems. This point is most easily understood in terms of the structure of the directory, explained in the remainder of this section.

The directory is a tree structured file in which each node of the tree is a record. If the record points to other records it is a directory (or sub-directory) node. The ends, or leaves, of the tree are descriptor nodes that contain a data description. A record that describes a file may be referred to as a "file" node--a special case of a descriptor node. Each descriptor node has a name--the name of the highest-level data container described. Names below this level are not known to the directory system.

Each record has a "node name" that is a character string without any blanks or other special characters. Each record also has a "path name" that is the concatenation of all the node names (with periods in between) of the directory records traversed in order to arrive at the selected node. Path names will be unique. Node names may be repeated.

It is expected that the higher nodes will correspond to organizations, projects and/or people, but the directory system will not assume this. The path name for a file's descriptor record is that file's "formal name". Since formal names are unwieldy, files will be referenced by "normal names". The request handler will prefix a path name to the normal name in order to derive the formal name.

Nodes that describe lists of files or files composed of subfiles will be specified at a later date.

6.3 File Numbers

Files may be referenced by any of three types of file numbers. First, a permanent file number (PFN) is associated with a physical file. It is assigned by the directory system and never changes even if the file is renamed. PFNs are never reused, even if a file is deleted. Second, the directory file number (DFN) is a pointer to the file node in the directory. It corresponds to a logical file. It is used in calls to the directory system, and to the storage manager when opening a file. Third is the local file number (LFN) assigned by the storage manager. It is meaningful only for open files, and is used in calls to the storage manager.

6.4 Calls from the Storage Manager

When the storage manager calls the directory system, it always supplies a DFN as a primary argument, and a pointer to an extent block (or partial block) as an optional argument. The directory system always returns a pointer to a complete, up-to-date extent block corresponding to that file. There are entries to get a map when opening a file, allocate more space within a file, and free up unused/unneeded space within a file.

6.5 Calls from the Request Handler

When the request handler calls the directory system it always supplies a directory pointer to be used as the "top" node. Zero is used to specify the real top. This is used to speed up directory searches. The second argument is the path name of the desired node. The directory system always returns a pointer to the up-to-date node. There are calls to search, create, delete, and modify directory records.

6.6 Datalanguage-Directory Interactions

Several datalanguage commands interact closely with the directory system. They are outlined here.

LOGOUT causes the system to forget everything that gets set up by the following commands.

ACCOUNT specifies a user's account number to the datacomputer.

ATTACH specifies a prefix to be used when converting normal names to full names. A user may be attached to several nodes at one time. If so, they are searched in order.

LOGIN combines LOGOUT, ACCOUNT, and ATTACH, in that order.

OPEN specifies a normal file name that is to be used when trying to identify a datacomputer name. A user may have several files open at once. If so, the options supplied by OPEN determine if the first match is accepted, if ambiguities result in an error, or if all matches will be processed.

CLOSE un-opens a file or all the user's open files.

6.7 Restrictions

Any node of the file directory may contain a restriction block. As directories are searched, the privilege bits from an entry for the user are ANDed together. Whenever a privilege is revoked (by setting its corresponding bit to zero) it is also revoked for all nodes below. If no restriction block is present, no restrictions are revoked. If a block exists, but no entry corresponds to the user, all privileges are revoked and the search is aborted.

Any entry in a privilege block may contain a password. If the user has supplied a password, entries with passwords are checked first. There may be multiple entries for a user with different passwords to allow a user to have different privileges at different times. If the password does not match, other entries are checked.

It is expected that most directories that correspond to projects or users will have privilege blocks. Normally these blocks will allow read only access to most data by most users and read/write access to users of the same group. Specific restrictions can be provided to restricted users or groups by providing the appropriate password entries.

To the datacomputer, a user is identified by a path name that corresponds to the first directory he is attached to. This will normally correspond to the group/person identification used by most time-sharing systems. The "*" convention for any node name will be necessary for convenient use of the restriction mechanism. "*" will be used to indicate all subordinate nodes.